NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTRUSION DETECTION IN REAL TIME IN A MULTI-NODE, MULTI-HOST ENVIRONMENT

by

Joseph D. Barrus

September 1997

Thesis Advisor: Neil Rowe

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE September 1997	1. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE INTRUSION DETECTION IN REAL TIME IN A MULTI-NODE, MULTI-HOST ENVIRONMENT	5. FUNDING NUMBERS
6. AUTHOR Barrus, Joseph D.	
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSOR/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense of the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

While there exist many tools and methods used to recognize intrusions into single system environments, there are few that can recognize and handle attacks in real time. This group is further reduced when adding the complexity of recognizing and handling intrusions occurring in a heterogeneous networked environment. The results of the thesis are an open architecture design for a real-time intrusion detection system to handle intrusions in a heterogeneous network and the system requirements, specifications, protocols and software module design to support an implementation of a system using this architecture. The architecture presented herein comprises a distributed system of autonomous agents that reside on the various hosts in a network. These agents communicate with each other in a coordinated effort to identify and respond to intrusions into the network by sending messages to each other detailing the identity and threat level of a potential or imminent attack. To quantify the threat level of an ongoing attack, this thesis also presents an alert level hierarchy based on the danger level and transferability of the threat to the various hosts within the network.

14. SUBJECT TERMS Intrusion Detection, Network Security,	15. NUMBER OF PAGES 92
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT **UNCLASSIFIED**	18. SECURITY CLASSIFICATION OF THIS PAGE **UNCLASSIFIED**	19. SECURITY CLASSIFICATION OF ABSTRACT **UNCLASSIFIED**	20. LIMITATION OF ABSTRACT **UL**

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-18

INTRUSION DETECTION IN REAL-TIME IN A MULTI-NODE, MULTI-HOST ENVIRONMENT

Joseph D. Barrus
B.S.M.E., Rice University, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

NPS Archive
1997.09
Barns, J.

~~N5~~
~~B24416~~
~~c.1~~

ABSTRACT

While there exist many tools and methods used to recognize intrusions into single system environments, there are few that can recognize and handle attacks in real time. This group is further reduced when adding the complexity of recognizing and handling intrusions occurring in a heterogeneous networked environment. The results of the thesis are an open architecture design for a real-time intrusion detection system to handle intrusions in a heterogeneous network and the system requirements, specifications, protocols and software module design to support an implementation of a system using this architecture. The architecture presented herein comprises a distributed system of autonomous agents that reside on the various hosts in a network. These agents communicate with each other in a coordinated effort to identify and respond to intrusions into the network by sending messages to each other detailing the identity and threat level of a potential or imminent attack. To quantify the threat level of an ongoing attack, this thesis also presents an alert level hierarchy based on the danger level and transferability of the threat to the various hosts within the network.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to thank Ron Broersma, my boss, for sponsoring me locally on my thesis. Without this sponsorship and his willingness to be my local advisor, I might not have found enough time to succeed.

I would also like to thank Duston Hayward of NRaD and Mantak Shing of Naval Postgraduate School for all their effort in providing the logistic support in getting all the proper routing of documents to and from both locations, the signatures needed and all the other bells and whistles required to get this thesis approved in time for September 1997 graduation.

And lastly, but certainly not least, I would like to thank my wife, Leslie, for proofing my thesis for grammar, fluidity and style. I needed that.

I. INTRODUCTION

A. VULNERABILITIES

The enormous growth of the Internet has not come without problems. Many more people are taking advantage of the rapid advances in network technology by connecting their systems and networks to the Internet. Their intentions are varied. They include the traditional exchange of information and ideas to support research, education, etc. But the most explosive growth has been in the commercial sector with companies advertising their products and services as well as placing their wares, services, etc. online for access and sale. While this is generally a good thing, it does greatly increase the vulnerability to attack of all systems on the Internet. There are more systems to attack as well as more systems from which to attack. These attacks take advantage of the flaws or omissions that exist within the various operating systems and software that run on the many hosts in the network.

Why these attacks exist in the first place can be explained by simple sociology. The computer network world is just an extension of our society and comes with the same variety of personalities. The difference here is that advanced technology can provide very sophisticated tools with which to do damage or to hide from recognition. The kind of damage that can occur includes theft of information and resources, malicious vandalism, violation of privacy, violation of national security, industrial espionage, etc. This is usually accomplished by a subversion of whatever existing security mechanisms are deployed on a host with the intent to execute operations or gain access to areas beyond the allowed authorization.

The various security mechanisms that exist are numerous. Each operating system usually comes with its own protections against access to certain files and processes. This is further varied by the different versions of these same operating systems which might provide slightly different implementations of the same mechanisms. Other security mechanisms include third party software that sits on top of the system kernel and provide additional services like monitoring the activity on the system looking for specific violations. The degree of protection that exists on any given host depends on how much time and effort has been put into applying these mechanisms. Often times, this requires much more time and effort than is desired by the various system administrators and many systems are left in a quite open and extremely vulnerable state. Many of the security mechanisms have a significant effect on the performance of a system and thus may not be used for this reason as well.

B. INTRUDER RECOGNITION

When an intruder attacks a system, the ideal response would be to stop his activity before he can do any damage or access sensitive information. This would require recognition of the attack as it takes place in real time. There are few automated methods to perform this recognition. Most real-time methods are applied manually by human observation by explicitly looking for the attack as a result of previous analysis or due to some triggering event of suspicious behavior, probably recognized by blind luck. The triggering event does not necessarily imply an intrusion, but is enough of an indicator for the system administrator to pay close attention. If further behavior continues to suggest an intrusion, the system administrator can confirm this via manual contact with the owner of the account from which the attack originates. He then can take action to halt the attack if necessary.

Recognition of an attack is further complicated in a networked environment. A single suspect behavior on a single host in a network may not warrant any serious action. However, repeated suspect behavior across several hosts in a network may indeed suggest an attack, with a response definitely warranted. This would be very difficult for a human to recognize.

C. THE CHALLENGE

The challenge is to automate this process in a networked environment in such a way that is effective, efficient, scaleable, lower in overhead, fault-tolerant, flexible, extensible and resistant to compromise. This is no small task. Some attempts at this have been tried with most methods concentrating on analyzing the network traffic at the single point it passes into the network. Certainly a worthwhile endeavor, but in and of itself, it may not be enough. This kind of approach does not scale well and may run into difficulty in dealing with large amounts of data possibly overlooking some attacks. It also does not address attacks coming from within the network. Network administrators must be concerned with the unauthorized behavior of their own users. As network traffic increases, many of these intrusion detection systems can suffer significant performance degradation. These systems also operate as a single entity offering only a single point of failure, thus being vulnerable to execution faults or failures, or being compromised by external sources.

Also, most current intrusion detection systems do not offer real-time response, but rather are used to make determinations that an attack did occur and that some action can be taken after the fact. This may be too late as the intended damage may already be done. Finally, these systems are not very flexible when it comes to extending the capability of the system to recognize the ever-growing pool of attack types. Often, this would require taking the system down and doing a complete reinstallation, thus leaving the entire network vulnerable while this is taking place.

This paper will address an approach that attempts to provide a solution to these specific issues. In general, this paper will propose a distributed architecture using autonomous agents for a system to monitor security related activity within a network. Each agent operates cooperatively yet independently of each other providing for better efficiency, real-time response and distribution of resources. This kind of architecture also provides significant advantages in dealing with the issues, such as scaleability, flexibility, extensibility, fault tolerance, resistance to compromise, etc. In addition, this thesis offers an object-oriented modular approach to the design of the agent software, helping to solve many of these issues, particularly extensibility and efficiency.

Chapter II discusses the intrusion detection domain, the methodologies and approaches to the various problems. It also discusses the requirements necessary for a proper intrusion detection system. Chapter III details specific intrusion detection systems previously offered as solutions and their advantages and disadvantages. Chapter IV proposes the new approach for a solution that is the subject of this thesis. Finally, Chapter V discusses the results of this approach and how it satisfies the requirements for a good network intrusion detection system and offers up some conclusive statements.

II. APPLICATION DOMAIN

A. INTRUSION DETECTION

An *intrusion attempt* can be defined as the "potential possibility of a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable." [Ref. 1] This kind of behavior obviously constitutes misuse and abuse of a computer system. An *intrusion detection system* can be defined as a system that "must identify, preferably in real time, unauthorized use, misuse, and abuse of computer systems." [Ref. 2]

Most intrusions can be classified into either one of two large groups: *Misuse* intrusions or *Anomaly* intrusions. Misuse attacks are well defined, previously documented attacks. Their behavior is known. Most systems that handle these types of attacks do so by trying to recognize the attack pattern as it occurs. Ideally, misuse detection systems can represent known attacks as a type of signature so that even variations of the same type of attack can be matched against this signature and recognized. Anomaly intrusions are recognized through deviations from the normal behavior of a system or user. Profiles of system or user behavior are defined and stored with subsequent behavior being compared to this profile. If it deviates beyond a certain threshold, a flag can be raised. Anomaly detection usually requires time to establish and maintain credible profiles and is more susceptible to false positives than misuse detection. Several approaches have been established to monitor both types of intrusions.

B. MISUSE DETECTION

Misuse detection mechanisms have used the following schemes to detect intrusions within a system or network.

1. Expert Systems

As the most commonly used method of detecting misuse intrusions, expert systems rely on a set of rules specifying conditions that must be met. If the full set of rules are met, an intrusion is detected. A major disadvantage of expert systems is that the rules set is non-sequential; that is, there is no ordering implied to the set of rules. This may make it difficult to positively identify an attack in those cases where they can only be determined by their ordering of events. Furthermore, expert systems also are only as good as the rules that have been placed in their rules base. If the rules base is not properly populated, then some intrusions may be missed.

2. Model Based Approaches

This method uses model definitions of misuse attacks and combines them with evidential reasoning to identify the occurrence of an attack. Certain activities are monitored and the system expects that these activities may imply a particular intrusion scenario is taking place. The model based approach comprises three main components. The *anticipator* predicts the next behavior in the active intrusion model based on the scenario model and passes this to the *planner*. The planner then determines the format this behavior will have in the audit trail. The *interpreter* then searches for a match of this behavior in the audit trail, continuing the search until there is enough evidence to flag an intrusion. This method can be very efficient in filtering through the audit trail data improving performance. Also, the evidentiary reasoning involved helps offset false positives. However, false positives can only be eliminated if the models absolutely do not mimic any normal behavior. Intrusions can also be overlooked if patterns are not easily recognizable.

3. Pattern Matching

This approach, which attempts to represent attacks as a pattern or signature able to be matched against the audit data, is a promising solution for real-time detection. As events occur, they can be translated into a pattern matched to known intrusion patterns until a full match is made. This kind of algorithm is efficient and has a low overhead, thus providing good basis for a real-time system. It also is system-independent since intrusions are represented as patterns rather than system-specific events or in different audit trail formats. It can handle multiple attack threads better as each thread can be processed independently and the results matched across each other to identify a pattern. However, this method only works with known attacks that can be represented as a well defined pattern, not a simple task to execute. Also, not all known intrusions can be translated into patterns, such as spoofing.

4. State Transition Analysis

This method analyzes data, identifying events that transition the system from one state to the next. The actual transition from one state to the next is protected by a condition guard. Not only does the event have to occur, but the condition must be met as well. Once the system has reached an "unsafe" state then it can take action. The placement of the guarded conditions helps to filter out non-intrusive situations. This approach is good for detecting cooperative attacks and attacks that span user sessions. A quantifiable advantage of this methodology with respect to a real-time system is that it is efficient at predicting the approach of an unsafe state of an attack, thus allowing for preemptive recourse.

5. Specification-Based Approaches

A new approach being developed by the Department of Computer Science, University of California, Davis, their idea is different than most traditional misuse detection methods, focusing on detecting the misuse of executable programs, particularly in privileged programs that run on UNIX. Traditional methods rely on previous knowledge about the vulnerabilities in these programs in order to recognize an attack taking advantage of those vulnerabilities. This new method takes an opposite approach. It recognizes attacks without requiring any prior knowledge about specific attacks taking advantage of vulnerabilities, which can be numerous. This method relies on a written specification based on the desired behavior of the program, the security policy of the system and security concepts. Then the actual behavior of the program is monitored for violations of this specification. This method is similar to anomaly detection, but can be used specifically to detect misuse situations. This has the obvious advantage of not having to keep up with documenting all the new types of attack signatures. It can be very useful in reducing the maintenance overhead of a system.

C. ANOMALOUS BEHAVIOR DETECTION

Detecting deviations from normal behavior of systems requires a different approach. Anomaly intrusion detection mechanisms makes use of the following schemes.

1. Statistical Analysis

Traditionally the most common approach, this method compares the current system behavior with previously defined profiles of the system or users. If the current profile of a user differs significantly from the pre-defined profile, then an intrusion may be taking place. Probability analysis can be applied to determine this threshold. This approach does not need to know about the security vulnerabilities of the system as it is only the deviation from normal behavior that raises a flag. In addition, this approach can adapt well to changing behavior of the system as newer statistical data on profiles can be merged into the historical profile data. However, this approach is susceptible to high false positive and false negative rates depending on how well the threshold levels are set. Also, it can be fooled by gradually teaching the system to accept inappropriate changes in behavior.

2. Predictive Pattern Generation or Rule-Based Detection

This approach assumes that anomalous behavior can be represented as a predictive pattern. The patterns are represented as rules describing events in a sequence that can predict the probability of subsequent events occurring. [Ref. 3] gives this example:

E1->E2->E3 => (E4 = 95%, E5 = 5%)

where, if event E1 is followed by E2 followed by E3, then the probability of E4 occurring is 95% and the probability of E5 occurring is 5%. This approach adds the additional aspect of ordering to the analysis, something was missing in the statistical approach. It also is very adaptive to changes because it can eliminate low quality patterns. Additionally, it is more efficient in terms of performance and is more resistant to surreptitious training.

3. Neural Networks

This approach attempts to predict the user's next action based on a previous window of actions or commands. The network is trained on a representative set of user commands or actions to determine a profile of that user. The network then attempts to match the actual user commands against the predicted command based on the stored profile. The sum of the incorrectly predicted events represents the deviation from normal behavior. This approach does not depend on any statistical nature of the user's activity and copes well with noisy or fuzzy data. However, this approach does have some drawbacks. The reliability of the system is directly dependent on the size of the window of previous actions or commands. Too narrow a window will result in a high rate of false positives and too high a window will introduce too much irrelevant data and may result in an increase in false negatives. Also, the topology of the network can only be determined by extensive trial and error. The network is also susceptible to being surreptitiously trained.

Each one of these methods offers a viable approach to detecting intrusions. However, no single method is good at detecting all intrusions. Most systems apply only one or two methods for detection, but it is clear here that it would be preferable for a system to take advantage of all these methods by applying the best method available for a specific intrusion type. This is particularly true when applying intrusion detection in a network of variant environments.

D. WHY DO WE NEED INTRUSION DETECTION?

Why do we need intrusion detection systems in the first place? Why can we not build completely secure systems that utilize strict authorization methods, cryptographic encoding schemes on data and very tight access controls to the data? Would this not be enough? The answer is really no. Sandaram [Ref. 1] offers the following reasoning in response to these questions:

1) It is not practical to build an entirely secure system. Bug-free software is a dream that will never happen, therefore our operating systems will be continuously subjected to potential violations due to buggy software. The increasing competitive commercial software market has made it the norm to release buggy software in order to be the first to market, so nobody is interested in spending the time to make bug-free software. There is no

practical way to control nor predict the kind of access violations that can result from buggy software.

2) Cryptography is not absolutely secure. There are ways to break codes. The more sophisticated and powerful the hardware gets, so goes the increase in sophistication of the tools for breaking these codes. It is just a matter of time.

3) You cannot adequately protect from inside abuse from those who rightfully have certain access privileges.

4) The more restrictive a system gets, the more inefficient the users get. Therefore, users and administrators are not always willing to put up with the difficulties associated with security.

Traditionally, before the explosive growth of networking, intrusion detection systems were host-based systems. Host-based systems only protect single systems and if multiple systems in a network needed protection, then this software had to be installed on and configured for each machine. In order to alleviate some of the problems with multiple installations and to catch network level intrusions, network-based systems were introduced. They usually are a single system on a separate host analyzing collective audit data from the various hosts or by directly filtering through network packets. Most all of these systems to date, both host-based and network-based, are large, single-module, monolithic systems. Thereby presenting some real difficulties,

1) They imposes a significant overhead on the system upon which they run and the systems being monitored. Quite a bit of processing power is required to read and analyze all the data that is produced. A single-user, moderately loaded SunOS 5.4 system can produce up to 100MB of audit data a day [Ref. 1], thus these systems require quite a bit of storage space as well. In addition, a certain amount of load is required to generate the data by all the systems being monitored. In addition, the intrusion detection system itself may produce its own aggregate data and would require additional storage space.

2) They are not very flexible to changes or additions. Adding additional capability to handle new types of intrusion is no small task. It usually would

require a re-compile and reinstallation of the entire system module. Not only is this difficult and time consuming, but it leaves the network vulnerable during this time.

3) They are not very efficient. Monitoring the entire network requires the work of a single CPU. As the network traffic increases, so does the data required to be analyzed, therefore requiring more CPU time

4) They do not scale. As mentioned above, as the network grows, so does the load on the system. It is possible that this could reach a critical point after which the system becomes inadequate and the network becomes vulnerable.

5) They are not fault-tolerant. Monolithic systems offer only a single point of failure. Disrupt that process and you can disable the security system for an entire network.

E. REQUIREMENTS ANALYSIS

Any good intrusion detection system in a network must meet certain requirements. The architecture and system design proposed in this thesis intend to satisfy the requirements. The following applies.

1. Requirement 1: Recognition

The system must recognize initial potentially suspect behavior (the triggering event). These events obviously equate to the lowest level of alertness; that is, the measure of the overall threat to the network and its components. These are events that may raise an eyebrow, but in and of themselves, may not represent anything sinister. Examples could be a series of unsuccessful logins. A user may have legitimately forgotten his password, triggering him to contact the system administrator for a new password (or look in his drawer where he wrote it down!). On the other hand, it might represent the first stage in a password guessing scheme. The system should respond by going into a low level alert and after some time-out period, return to normal if nothing further suspect occurs. The idea is that this triggering event should result in some low level monitoring of activity on some or all hosts and that it is in the analysis of the results of this monitoring that determines the next state of alertness.

The system must also be able to recognize an attack taking place based on a triggering event of a distributed nature in the network. In other words, the attack may start out as a series of innocuous events on single hosts, but it is their combination that raises a flag. For example, an attack may comprise a series of single password guesses on each host in a network. The single guess on a single host is not enough to warrant attention, but a series of single guesses from the same source across several hosts within a short time frame should. The system should be able to recognize an attack of this type.

2. Requirement 2: Escalating Behavior

The system must recognize escalating suspicious behavior on a single host or across hosts. At each level of alertness, some further activity or lack of it should result in a change in the state of alertness. At its simplest, the system at any time should either be at the lowest level of alert, escalating, or returning to the lowest level. More sophisticated representations of an alert level hierarchy, such as a tree or partial ordering could be used to better define variations in alertness, but at its simplest, a linear representation should be provided. System responses should correlate to some combination of alertness level, system events or activity, and attack type and stage of attack.

The system must be able to recognize that continuing suspicious behavior is relevant to a previously established attack thread. An attack thread is defined as the continuous attack stimulus and system response events that are deemed related and occurring in order. The system must be able to recognize and establish this relationship in order to build and maintain a thread. The system then must be able to recognize the point at which the escalating behavior warrants an upgrade in the alertness level. It may or may not be necessary to try to follow an attack thread across hosts, but some recognition is necessary. It might very well be that the system upgrades the level of alertness when the number of concurrent attack threads of the same type across hosts reaches a certain threshold. Similarly, the system must be able to recognize that further related activity, or lack thereof, indicates the attack has subsided and warrants a de-escalation of alertness.

3. Requirement 3: Communication

The system components on various hosts must communicate with each other, apprising them of a change in the level of alertness, whether escalating or de-escalating. In order for the system to recognize escalating suspicious behavior in a networked environment, any components that may be distributed across hosts must be kept aware of activity occurring on other hosts. This is important because it is behavior that exists across hosts that might indicate an escalation in an attack. An attack might be designed to perform small innocuous attacks across hosts in a round-robin sort of way to avoid detection. This system must try to recognize such an attempt and make sure that the entire system is aware of it. An escalation in alertness will likely result in changes in the system response and each distributed component must be made aware of it.

4. Requirement 4: Response

The system components must take the correct action that corresponds with a level of alertness, responding appropriately to changing levels of alertness. A triggering event may result only in turning on some additional logging or additional monitoring activity. If the attack thread fails to continue, then these activities can be turned off. If, however, it is determined the attack is escalating, then the next appropriate response might be to notify the administrator (email, page, phone call, etc.). And, if the attack reaches a dangerous level, then the next step in the response might be to kill the processes, disable an account, or turn off the network access port. In a technically sophisticated system, some sort of tracking response could be initiated to find and identify the intruder. Additionally, a good real-time system should allow for preemptive response when available. Preemptive here means having seized the initiative to thwart an impending attack. A determination of an existing or potential attack on a system in a network should be the catalyst for other nodes in the network being notified, resulting in awareness of the possibility that such an attack may soon be attempted. Thus forewarned, immediate preemptive measures to minimize the damage that may occur if the attack does take place can be effected.

5. Requirement 5: Manual Control

The system must allow for manual intervention in the changing of alertness. This is particularly important to allow for the de-escalation of alertness after the threat is eliminated. At any given time, the system must allow for a system administrator to manually override any state or ongoing operation. In fact, it may be that the appropriate system response to a level of attack is to wait for a human response. In which case, the administrator would be manually setting the level. Certainly, a "delete attack thread" or "reset" function would be desirable.

6. Requirement 6: Adaptability

The system must be highly adaptable to be enable quick responses to the ever changing methods of attack. This requirement applies to the overall design and architecture as well as in its operation. Initially, the system must be designed well enough to be flexible to add additional attack profiles and responses. New avenues of attack are continuously being applied as new software is added or upgraded to a system or the invaders get more creative. Once an attack can be recognized and documented, the system should allow it to be added with minimal effort and automatically applied.

Other changes to the system, such as configuration information, and even software upgrades, should be able to be automatically applied by propagating these changes throughout the network. This is necessary to avoid the difficulty of manually applying these changes to each host one at a time and to allow for rapid adaptation of the system to handle new forms of attack.

As the system evolves, it could possibly become sophisticated enough to apply some sort of artificial intelligence to dynamically identify and learn new forms of attack as they happen, quickly classify them, and automatically propagate the changes throughout the network. The system should keep a record of previous attacks, and when some new suspicious behavior occurs that is similar to a previous intrusion, a flag can be raised. Previous attacks can act as a knowledge base from which to learn about and identify potential new forms of attack, providing the ideal situation for minimal human intervention. However, to what level this can be applied is left to future endeavors. Initially, the system should allow for manual application of changes.

7. Requirement 7: Concurrency

The system must be able to handle multiple, concurrent attack threads. System effectiveness requires an ability to handle multiple attack threads occurring at the same time spawning new component processes if necessary. Attacks may be coming from the same or different sources and may require different types of response. Multiple attacks emanating from the same source may in fact indicate a rapid escalation in the alertness level. The actual number of concurrent attacks may in itself require different types of response, i.e., alerting the administrator or even major events such as system shutdown or network cutoff. The system must be able to identify the true source of an attack and relate the attack to other separate attacks coming from the same source, as the aggregate of these attacks may represent a coordinated attack.

8. Requirement 8: Scaleability

The system must be able to scale as the network grows. The system must be able to handle any number of hosts and to automatically adapt to additional hosts being added. The architecture in itself should satisfy this requirement.

9. Requirement 9: Resistance to Compromise of Intrusion Software Itself

The system must be able to protect itself from unauthorized use or attack. Not only should the system be performing its mission by protecting systems from outside attack, but it should be able to protect itself from these same attacks. One of the first things a serious intruder may do is to attempt to disable the security software, much like a burglar would disable a home security alarm. Therefore, certain things must be done, such as encrypting the broadcast messages, protecting from overload, etc.

10. Requirement 10: Efficiency and Reliability

The system should be efficient, reliable, resistant to degradation and run in a heterogeneous platform environment. These are general terms that apply directly to the real-time aspects of the system. In order to be effective in real time, the system must perform well and quickly, be fault-tolerant and resistant to being overloaded.

11. Requirement 11: Extensibility

The system must be extensible to enable integration of additional intrusion detection capabilities as they become available, and to incorporate new attack types as they are discovered. Possibly the best source for becoming informed of new attack types is through the Computer Emergency Response Team (CERT) Coordination Center. This group issues periodic bulletins about new attack types and their methods. They also provide information about how to handle them which would prove useful in programming a response.

12. Requirement 12: Flexibility

The system must be flexible in the establishment of security policy and management across hosts. Many security systems are underutilized because of the constraints placed upon the system administrators and users. If the solution makes it hard to use the system, then the system does not get used. An intrusion detection system could fall into this trap, particularly if policy is set centrally. Many networks cross organizational boundaries and a centralized approach may not be enforceable. Therefore, a network intrusion detection system must, at least, allow for decentralized control and the setting of security policy amongst the hosts in the network.

F. SUMMARY

This is a fairly demanding set of requirements, that should result in a robust, efficient and successful network intrusion detection system. In reality, it is nearly impossible to capture the complete set, as much is still unknown about the nature of intrusions and their detection, particularly since the set of intrusions itself is not static, but growing every day. Many have attempted solutions to satisfy some or all of these requirements utilizing various methods. The next chapter will present some of these solutions, pointing out where they have succeeded and failed meet these requirements.

III. PREVIOUS SOLUTIONS

There have been several attempts to satisfy some or all these requirements for an network intrusion detection system. They all have merits but few have been able to satisfy all these requirements.

A. NETWORK SECURITY MODEL (NSM) OR NETWORK INTRUSION DETECTOR (NID)

This system is a LAN based system, consisting of a separate host sitting passively on a broadcast network monitoring network packets. There are several advantages to this system with regard to intrusion detection. The machine has immediate access to the network rather than through audit files, which has a second added benefit of not burdening the hosts with generating these audit files. Because it sits passively on the network, it can be invisible, therefore unable to be accessed by an intruder for attack itself. Also, since it incorporates data from a common, standardized communication layer, it can be used in a heterogeneous network. Finally, it can better recognize attacks on the network itself, something that would be difficult for host-based systems to do. On the other hand, this system only works in broadcast LANs and is not applicable to larger wide-area network topologies. It also will not scale well. If the network grows or the amount of network traffic increases, performance will be affected.

B. NEXT GENERATION INTRUSION-DETECTION EXPERT SYSTEM (NIDES)

NIDES is an extension of the single host-based system IDES. The hosts in a network transfer their audit files to a separate machine for central storage and processing with audit trails then subjected to analysis by a statistical anomaly detector for the detection of anomalous behavior and a rule-based expert system for detection of misuse. This kind of system has the advantage of looking for both anomalous and misuse attacks. However, this system also does not scale well as the machine hosting the NIDES system could be overburdened. They also do not monitor any network traffic.

C. DOROTHY DENNING'S GENERIC INTRUSION DETECTION MODEL

Denning [Ref. 3] developed a generalized model of an intrusion detection system. She generalized the detection approach so that it was independent of the system, type of input and type of intrusions. This model consists of several modules performing specific tasks. Figure 1 demonstrates. The system contains an Activity Profile which maintains information about the system's behavior and performs anomaly detection. The Rule Set maintains the rules and performs the misuse detection. The Event Generator is the module providing the input (events) regardless of whether their source are audit files, network packets, etc. This model took the step to generalize the nature of an intrusion detection system, thus providing the basis for systems that could be platform independent. Many intrusion detection systems have used this model as their basis, including NIDES. Different techniques and methods can be applied to any one of these modules depending on the actual implementation needed for the system or on other independent criteria. However, not all methods fit well into this type of model, such as neural networks or model-based approaches.

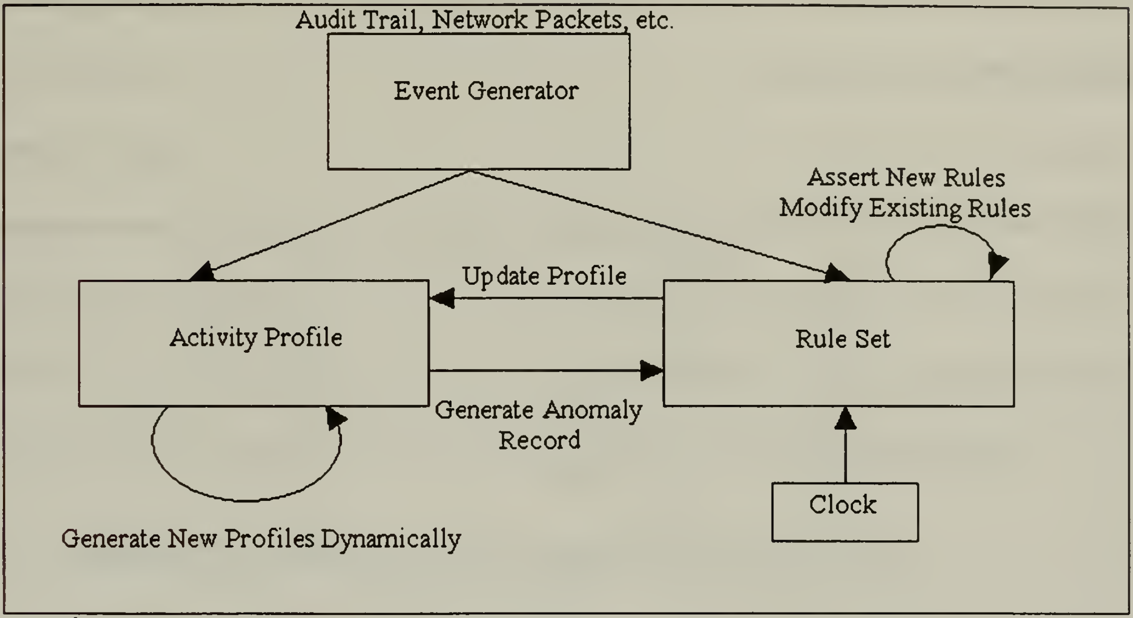

Figure 1. Dorothy Denning's Generic Intrusion Detection Model. After Ref. [3]

D. ADAPTIVE INTRUSION DETECTION SYSTEM (AID)

Utilizing a client-server architecture to communicate from a central intrusion detection monitor to various server agents sitting on hosts in a network, each server agent translates generated audit data into a platform-independent format, thus enabling applicability in a heterogeneous network. The manager performs the security administration by polling the various server agents for audit data, passing it to an expert system for analysis and returning any decisions made. This system has the advantages of working in a heterogeneous network and supports real-time decision making and response. It can scale moderately well as the agents can be distributed, but the single host detection system might not. Additionally, it only monitors for misuse and ignores anomaly detection, although that portion is under consideration for future development. Figure 2 outlines the architecture.

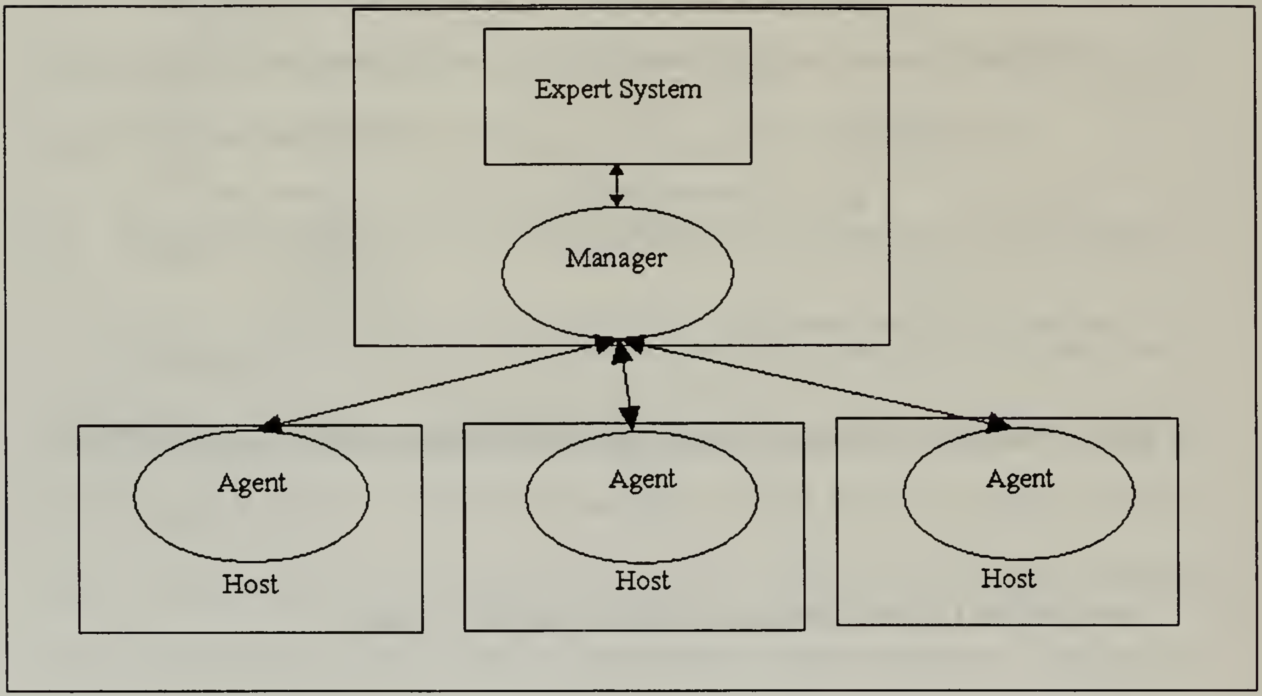

Figure 2. Adaptive Intrusion Detection System. After Ref. [4]

E. DISTRIBUTED RECOGNITION AND ACCOUNTABILITY (DRA)

Not a system, but a technique for identifying the source (user) of attacks throughout the network as proposed by Ko et al. [Ref. 5], this particular method can be very useful in identifying cooperative attacks. This technique requires a centralized expert system to apply the algorithm to audit files sent there from various hosts. This could be very useful in an architecture, as proposed in this thesis. Since it might be difficult, if not impossible, for individual hosts to recognize cooperative attacks, a separate system running the algorithm could be responsible for seeking these out. Then it could message the affected hosts with an alert message. This algorithm is designed to observe the ways an individual moves around in the network taking into account the changing of use names to hide his identity. This is the distributed recognition part. The second part is to associate all instances of this user by tying them together with a common identifier. This is the accountability part. This identifier can be added as part of the alert message to aid in keeping track of the activity of this user throughout the network.

F. DISTRIBUTED INTRUSION DETECTION SYSTEM (DIDS)

Mukherjee et al. [Ref. 6] proposed a distributed system, which basically is an extension of NSM or NID to a wide-area heterogeneous network topology. One of the first to take into account the distributed nature of some attacks across multiple hosts in a wide-area network, it also is one of the first to take extensibility into account in the architecture and to distribute the load across the various hosts.

This architecture essentially distributes monitoring activity amongst the various hosts. These monitors send "reports" of suspicious activity to a centralized *director* for analysis by an expert system. In addition, the LAN monitor sends its own "reports" to the director. The director can analyze the collective data looking for cooperative attacks. This system has the advantage of distributing some of its load across systems and is also extensible as additional monitors or tools can be added to the configuration as long as the adhere to a standard communication protocol. It is somewhat scaleable, as new monitors can be added as new hosts are added, but the director could possibly be overwhelmed with data, thus limiting its scaleability. It is possible that multiple directors could be added to mitigate this load. However, this system does not provide for real-time response to detected intrusions, but its extensibility could allow for the addition of incident handling tools to provide for real-time response. Also, the separation of activities, i.e., monitoring, analysis, response, throughout the network away from the hosts upon which an attack may be occurring may delay such response to the point where it might be too late. Figure 3 shows a diagram of this architecture.

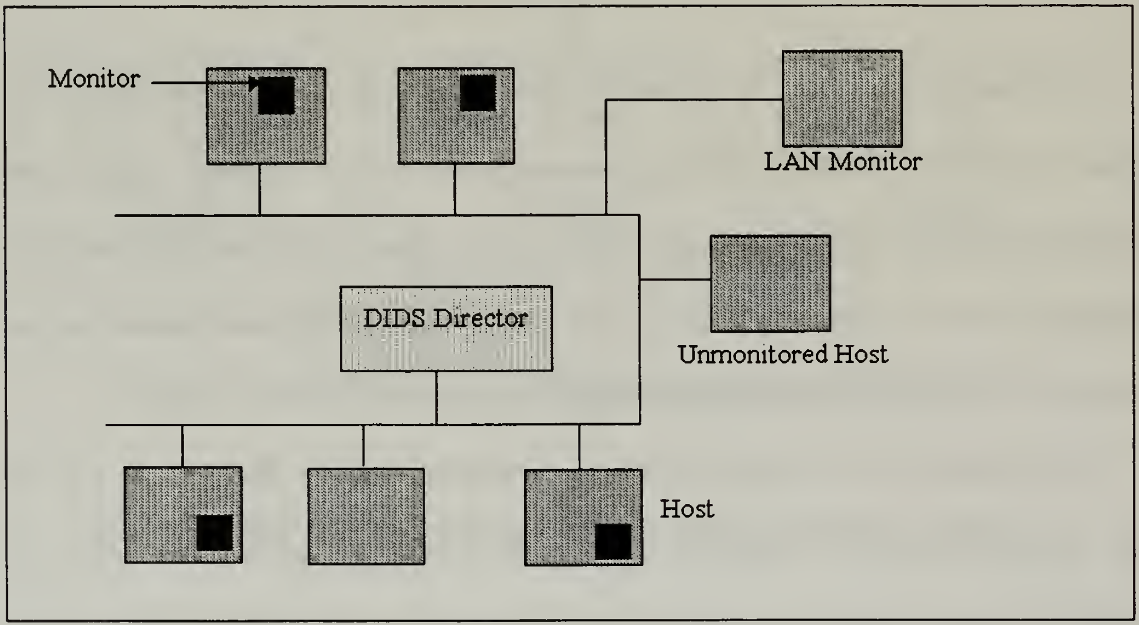

Figure 3. Distributed Intrusion Detection System. After Ref. [6]

G. GRAPH BASED INTRUSION DETECTION SYSTEM (GRIDS)

The Department of Computer Science at University of California, Davis has proposed an interesting approach to an intrusion detection system. [Ref. 7] This system was specifically designed to detect intrusions in large networks. Its intention was to directly address the scaling issue. It focuses on detecting network-based attacks like worms, sweeps and coordinated attacks, but can detect intrusions on individual hosts. The key to solving the scaling issue is its hierarchical architecture. Figure 4 outlines the architecture.

GRIDS represents network attacks as a connected graph where the nodes in the graph are the hosts or departments and the edges represent the activity between them. Each graph represents a connected set of events on the network. The nodes and edges have attributes that contain additional information about that attack. As the attack continues, the graph grows until some threshold is met; that is, either the graph has reached a particular determining event or has reached a determining size, indicating an attack. Different kinds of graphs are needed to represent differing types of attacks and are constructed from rule sets that specify how these graphs are built and what the thresholds are. The architecture consists of a hierarchy of departments, with each department containing hosts and/or departments. Additionally, there is a software manager which manages the hierarchy, and a graph engine which maintains the graphs. Each host contains a data source and a module controller. The data source are modules that monitor activity on a host or in the network, and can be anything that supplies data about intrusions, including host-based intrusion detection systems, network sniffers, LAN monitors or any other security tool providing data about a possible attack. The data source sends information about an intrusive activity to the graph engine in its department in the form of a node or edge on a graph. The graph engine makes a determination, using a set of rules, to include it in an existing graph or form a new one. The graph engine then sends summaries of those graphs to the graph engine of its parent department which subsequently repeats the process, but maintaining a graph at a coarser resolution. This system has the advantage of being scaleable, extensible and flexible. It handles detection of coordinated attacks well.

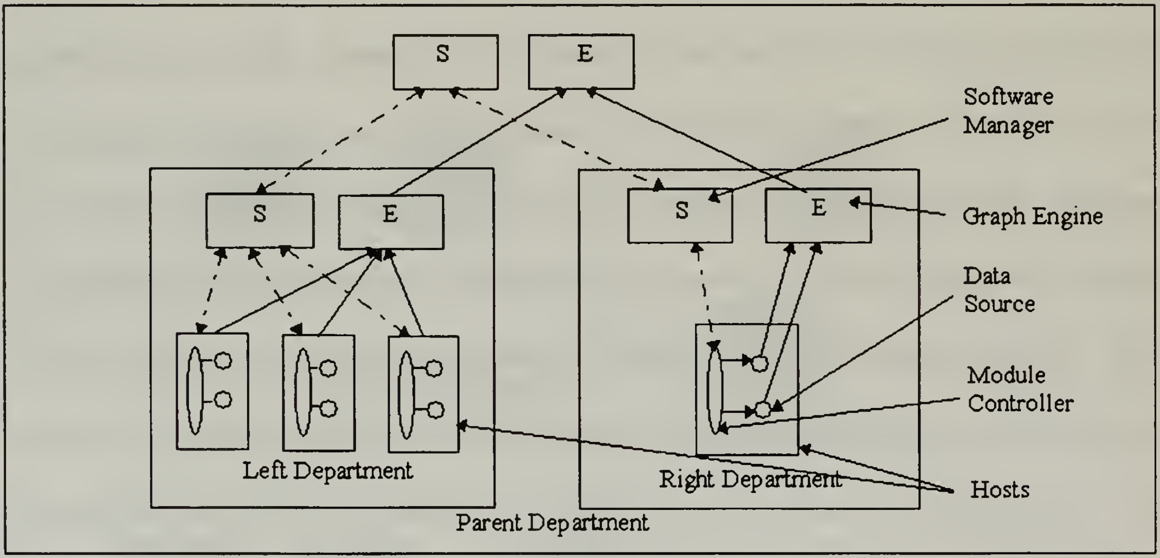

Figure 4. Graph Based Intrusion Detection System. After Ref. [7]

H. CROSBIE AND SPAFFORD'S AUTONOMOUS AGENTS

Crosbie and Spafford of the COAST project at Purdue University propose an system that offers the most promise. [Ref. 8] This approach is the one that most closely resembles the approach offered in this thesis, although there are significant differences. However, the main thrust of the architecture is the same. The system is distributed across the network via a system of autonomous agents cooperating with each other to detect intrusions within the network.

The advantages offered are very much the same as offered with the architecture described in this thesis, mainly scaleability, extensibility, efficiency, fault tolerance and resistance to degradation. However, this system focuses on delegating the functionality separately to the various agents. Each agent is only responsible for monitoring a small subset of activity, but at the network level. The agents look directly at the network level rather than for behavior on a specific host. So the load is definitely distributed here. This system also differs in that it adds a learning element. These agents must be trained to know how to seek out anomalous behavior in the network, so a training module is added to the architecture. This type of training could easily be added to the system proposed in this thesis as well. This system also provides for cooperative messaging between agents and carries the idea of varying levels of suspicion.

It differs from other intrusion detectors that work at the network level in that it operates on a wide-area network using TCP/IP protocol as opposed to most that work only in local area networks using a broadcast protocol. However, it appears this system is extensible enough to permit other types of monitoring to be added. Although this system claims fault tolerance; that is, the agent can be restarted at any time retaining state, this is not entirely true. While the state can be restored, the system is vulnerable during the down time. That particular functionality is not being performed throughout the system. True fault tolerance is achieved through redundancy in a real-time system and should approach no degradation if a fault occurs. Figure 5 below gives an overview of this architecture.

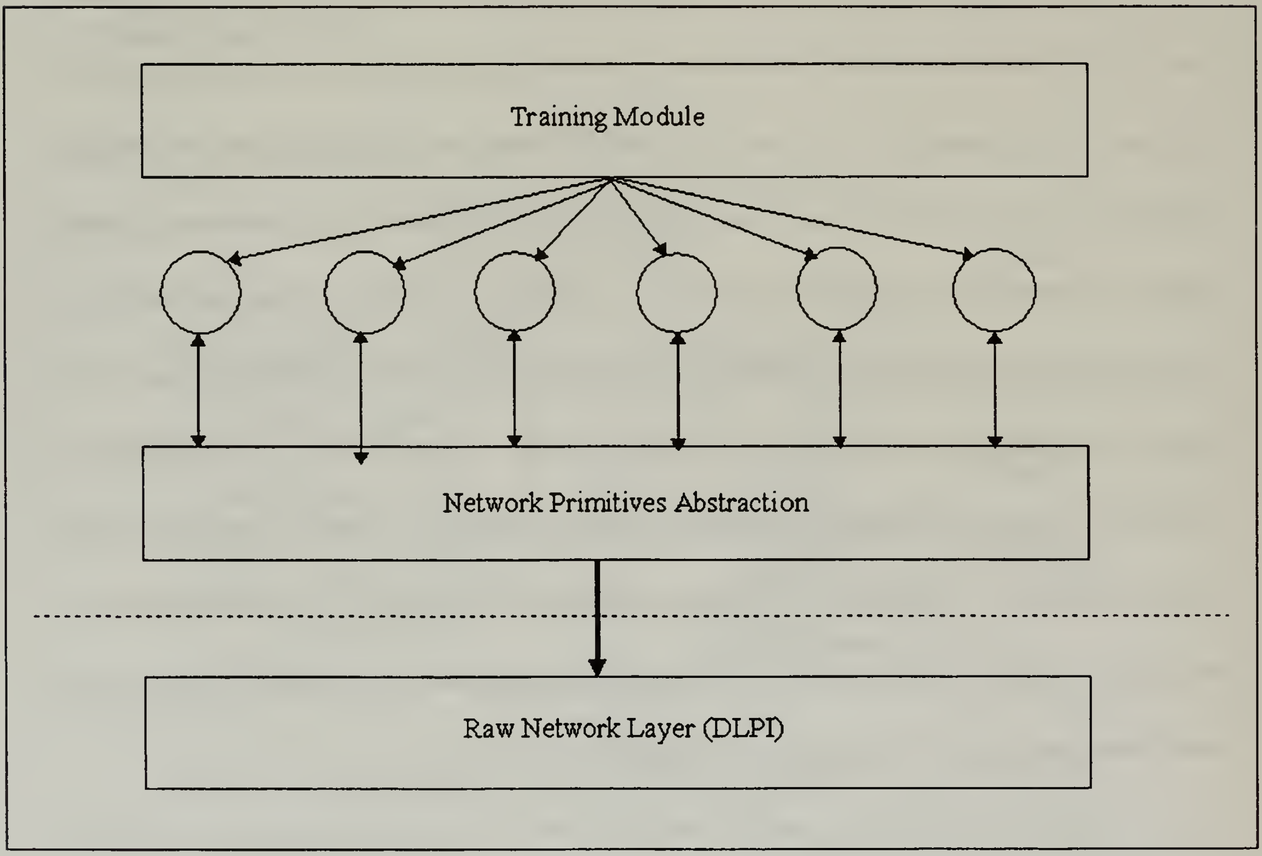

Figure 5. Crosbie and Spafford's Autonomous Agents System. After Ref. [8]

I. SUMMARY

This chapter has presented some past and current work in the attempt to build a robust yet successful intrusion detection system. Most have been moderately complete in meeting the requirements for a successful intrusion detection system. The next chapter will outline my solution: The Distributed Autonomous Agent Network Intrusion Detection and Response System.

IV. ARCHITECTURE AND DESIGN

A. A PROPOSED SYSTEM ARCHITECTURE

Figure 6 below is a simplified diagram of the proposed system architecture. Independent, autonomous agents sit atop various host machines in a network, communicating with each other via a defined protocol over a TCP/IP network.

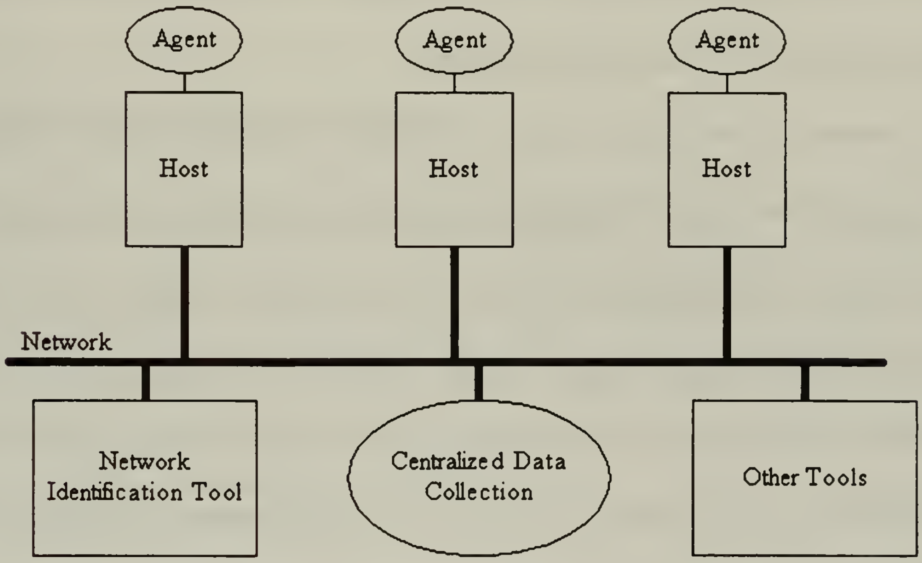

Figure 6. Distributed Autonomous Agent Network Intrusion Detection and Response System

The hosts are definite sites on the network, while the tools and centralized data collection may or may not reside on a network site. If they indeed are hosted on a network site, then the hosts upon which they reside would benefit from having an agent run there as well.

The agents are the main components of the system. They are processes that run on each host, monitor that host for intrusive activity and communicate with each other in a coordinated response to this activity. They are written and compiled specifically for the platform upon which they sit. They all share the same software module design but the actual implementations of the various methods are written to deal with intrusions as they occur within their particular environment, that is the operating system, application software, etc. of that particular platform. These agents run independently of each other. Their design is detailed in Section 6 of this chapter.

The Network Identification Tool is a separate mechanism used to recognize coordinated attacks that might be distributed throughout the network. It would be used to attempt to recognize that seemingly different users are actually coming from the same source, but may be attempting to masquerade their attack by distributing it across multiple sessions and/or user accounts. Alert messages generated from host monitors may be filtered through this tool to attempt to associate it with an ongoing coordinated attack. This is further discussed in Section 5 of this chapter.

Centralized data collection provides a central location where certain activities on various hosts can be recorded and accessed by any host in the network allowing for the recognition of those kind of attacks that are distributed in nature. Any host can act as the monitor for these types of attacks.

There are many other tools that exist in the intrusion detection world that could be useful if combined with the distributed monitoring on the hosts. LAN monitors, expert systems, intruder recognition and accountability, and intruder tracing tools are examples of some that could be very useful as augmentations to basic host monitoring infrastructure. These tools must work within the constraints of the architecture, mainly by utilizing the common communication mechanisms involved. This may mean some modifications to build an API to the communication layer.

The rest of this chapter will specify a system utilizing this architecture focusing on the attack hierarchy, alert level hierarchy, agent design and communication between agents.

B. FUNCTIONAL SPECIFICATION

This thesis will describe the design of a based on Figure 6. In doing so, the proposal contained herein will hopefully satisfy the requirements of a good intrusion detection system as described in the Requirements Analysis section of this thesis.

1. Attack Taxonomy

To make an intrusion detection system more flexible and extensible, it is desirable to establish an attack classification hierarchy. This accomplishes several things:

1) By abstracting details of attacks out of a higher layer of classification, generalization can be applied at the higher levels and the details left to the lower levels. This allows for a classification of attacks that can be applied across various platforms, where the details of how a particular attack's behavior on differing systems can be left to the lower levels.

2) Classification on type of attack rather than the techniques for monitoring them (the traditional way) allows for better recognition of these attacks from their manifestations in the audit records.

3) Generalizing common traits of attacks provides a basis (process of elimination) for more efficient algorithms to identify types of attacks.

4) Classification allows for better conceptualization and understanding of attacks and how they are interrelated.

5) Classification provides for better extensibility of the system by making it easier to add new attack types to the hierarchy, particularly when the behavior of a known attack varies slightly as intruders try new avenues. This way new behaviors can be added to the classification without losing or confusing previously defined information.

It is generally recognized that all intrusions fall into either one of two basic high level categories. They are Misuse and Anomalous Behavior. Misuse comprises attacks that are already known and whose behavior can be specified. These types of attacks are ones that have been previously recognized and documented. Therefore, recognizing the existence of an attack would involve simply comparing current behavior against a set of pre-defined attack scenarios. If a match occurs, then you have a likely attack. Matching behavior occurring early in the scenario would indicate a potential, but maybe not absolute, attack. As the attack continues to match as it progresses through later stages of the scenario, the more likely this type of attack is occurring. System responses can occur through various stages of this matching process.

Misuse intrusions can be categorized by type of attack. The following represents some of the category hierarchy:

1) Attempted break-in

2) Masquerade attack

3) Penetration of the security control system

4) Leakage

5) Denial of Service

6) Malicious use

Sandeep Kumar [Ref. 9] offers a very good classification scheme. Intrusions can be represented by an event or series of events. It is the relationship of these events to one another that provides the basis to recognize differing attack types. He offers four higher level classifications of attack types under which the above types can be placed:

1) Existence of Changes in System State: This type concentrates on the existence of events that vary the state of the system. This type usually involves examining system resources, such as the file system, for particular changes in state. Examples may be a change in permissions on protected files, the existence of particular files, or changes in the content or format of particular files. Many of the intrusions that are not recorded in audit trails can be classified here.

2) Sequential Patterns of Events: The specific sequence of events may specify a possible intrusion. Kumar further breaks this category down to subtypes, Interval and Duration. The Interval subtype covers events that happen within a specified time interval. The Duration subtype covers events that exist or occur for not more or not less than a specified interval of time. An example of an Interval subtype would be race condition attacks and an example of a Duration subtype would be a password guessing attack (repeated login failures within a specified time).

3) Regular Expression Patterns of Events: This type covers attacks that involve events that must occur jointly, but not in any specific order. For example event A happened AND event B happened AND event C happened, but not necessarily in any particular order.

4) Other Patterns: This is the "catchall" classification of attack patterns that do not fall in the above top level categories. Kumar suggests these further sub-classifications.

- Patterns that require embedded negation: This covers attacks that represented by the failure of an event to occur when expected.

- Patterns that involve generalized selection: A match on an attack occurs if a selected subset of events occur out of a specified superset of events, i.e., that any *x*-2 events occurred out of a set of *x* events.

Care must be taken to handle multiple inheritance issues. For example, an attack may comprise traits from both Sequence and RE Pattern types. Event A and Event B must happen in a certain sequence, but may be meaningless unless Event C happened at some time before or after. One should attempt to classify attacks of this nature under the most appropriate hierarchy, but may actually be classified under both to enable more efficient and responsive algorithms. On the other hand, the occurrence of multiple inheritance strongly suggests a poor hierarchy, in which case, the above suggested classification may not be particularly useful. It may be more appropriate to use the less general higher-level classifications than proposed by Kumar.

Anomalous Behavior describes those kinds of attacks that may not fall into any know misuse category, but can be detected by use of the system resources that fall out of the ordinary. Usually, this kind of behavior is measured via statistical methods that capture information about a user on a system and develops a profile of that user's behavior. This requires the existence of a profile generator which can continuously observe and update a user's profile. When behavior is observed that significantly deviates from the normal behavior, then the system can respond accordingly. Some threshold value must be established that would represent the probability of an attack occurring based upon the extent of the deviation from the norm of a behavior.

Anomalous intrusions can be categorized on types of behavior. Sandeep Kumar [Ref. 9] offers the following representation of some of the category hierarchy:

1) Activity intensity rate: The measure of the rate at which an activity is progressing. This can be used to detect unusual bursts in behavior deviating from the norm for a profile.

2) Audit record distribution of particular activities: Measures the distribution of activities over the entire system usage. Looks unusual clustering of activity types.

3) Categorical distribution of activities: Measures the distribution of a particular activity over categories. For example, it may measure the relative usage of a particular piece of software.

4) Ordinal measurable activities: Measures activity quantifiable in numeric form, such as the amount of CPU time used by a user.

Figure 7 below shows a graphical representation of the attack hierarchy. The forks below the rectangles represent inheritance.

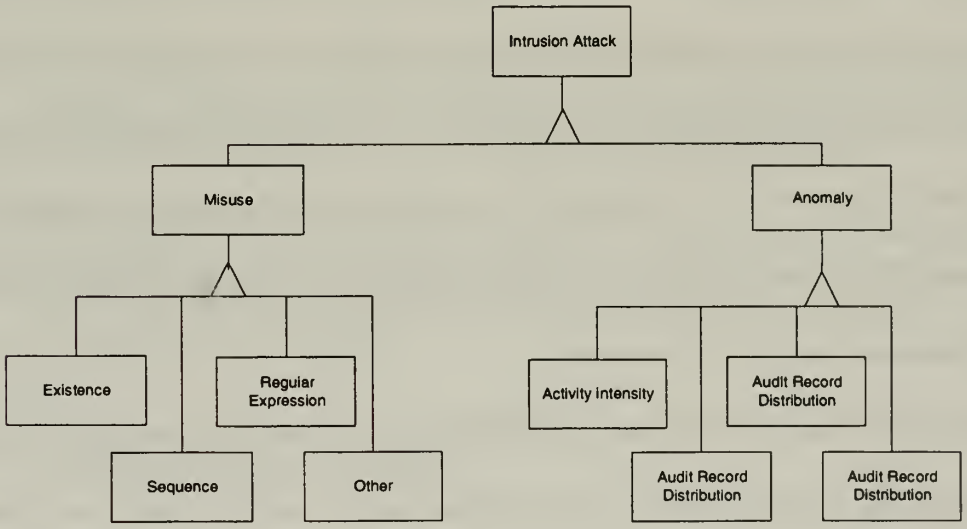

Figure 7. Intrusion Attack Class Hierarchy.

2. Alert level hierarchy

Establishing the seriousness of potential and ongoing attacks in a network requires a little more thought than just a simple hierarchy based on the danger level of each attack. While a potential attack is obviously relevant to the machine upon which it occurs, it may not be relevant to all the other machines in its network environment. A hierarchy of alert levels can be used to measure this relevancy. The alert level represents, more or less, the relative importance of notifying other agents on other computers in a network of a particular attack scenario taking place on a single node in the same network so that these other agents can take preemptive or reactive measures. In this way, the agents in the system can cooperate with each other in recognizing and handling an attack. There appear to be two main factors which can be used to quantify the alert level of a particular stage in an attack; that is, danger and transferability.

The danger can be defined as the potential damage that can occur if the attack at a particular stage is allowed to continue, and is measured by comparing the kind and stage of an attack against the frequency of false positives. The transferability can be defined as the applicability of an attack in other nodes in a network, and assessed by comparing the kind of attack against the existence of similar operating environments (OS and software in which the attack occurs) in the network.

The danger and transferability values can be defined independently for each identified stage in a particular attack. These values may or may not change over various stages of an attack. Their product can be used to determine the Alert Level of an attack in a network, and Appropriate system responses can be broken up and distributed across these Alert Levels. One of these responses is the notification to other machines in the network of an attack so that they can take an appropriate response or ignore it. Other responses may be to notify an administrator, start logging certain activities, or taking action to halt the attack.

The formula for determining the value of the Alert Level will be thus:

*Alert Level := Danger * Transferability*

Danger values are:

1) (1) Minimal: The event cannot be deterministically associated with a known attack, but could be an early stage of some known attack profile or could be the early stage of some anomalous behavior.

2) (2) Cautionary: The event indicates a secondary or later stage of a known attack, but one that still cannot be determined to be part of a known attack. It may also be the result of an accumulation of anomalous behavior having increased from the minimal stage, but not enough for serious concern or the event does determine a known attack, but the potential damage of the attack is minimal.

3) (3) Noticeable: The event definitely identifies a known attack with moderate damage potential or the accumulation of anomalous behavior has reached a level where it is likely that some hostile action is taking place.

4) (4) Serious: The event definitely identifies a known attack with high damage potential or the accumulation of anomalous behavior has reached a point that it is either disrupting service or is highly indicative that an attack is occurring with serious consequences.

5) (5) Catastrophic: The event identifies a known attack that, if left to continue, may result in catastrophic losses.

Transferability values are:

1) (1) None - The attack or potential attack is occurring in an environment unique in the network. There are no other environments like it that could see a similar attack or be part of the same attack. In other words, there are no other similar operating systems, like software in which the attack is occurring, or a combination of both.

2) (2) Partial - The attack is occurring in a common operating environment or common piece of software, but the version of which may be different enough that applicability may be questionable. More simply stated, the vulnerability to a particular attack may be due to a deficiency in a particular version of operating system or software.

3) (3) Full - The attack is occurring in a operating environment or piece of software that is common across the network. Other systems in the network are definitely vulnerable to the same attack.

Taking the product of Danger and Transferability results in a range of Alert Level values from one to 15. The system response to a particular attack at some stage of the attack can be tied directly to this value. For example, if the system recognizes a particular attack, it will determine the danger and transferability values applicable to that stage of the attack, find their product, and respond in a manner whose definition is tied to the value of the product. It is possible there could eventually be as many different responses as there are possible values as the system becomes more sophisticated. In fact a neural net could be trained to assign better, non-integer values to Danger and Transferability resulting in more precise Alert Level values that better reflect expert opinion as to what the proper Alert Levels should be at varying stages of an attack. However, initially, there will likely be less variation as the response will be tied to a discrete range of values. In general, the actual system response will be determined by the type of attack in addition to the alert level value. Therefore, the only response that can absolutely be generalized is the need to notify other nodes in the network. So, the system response can be described as follows:

The following are responses assigned to discrete Alert Level value ranges and the thresholds are arbitrarily defined based on intuition. Training and experience can be applied over time to determine better thresholds and responses.

1) (1-5): no notification to other nodes is necessary.

2) (6-10): Partial notification to other nodes is performed. Notification is sent to only the nodes that are determined to be relevant or where the relevancy is non-deterministic. This may result in notification of all nodes, particularly in the case where the danger and transferability values are both three. Full notification may also result if the restricted set of nodes cannot be determined from the node registry.

3) (11-15): Full notification to other nodes is performed. The only time values can fall into this range are when they apply to all nodes in the network and the danger level is at least serious.

Another consideration in defining the level of alertness in the system is the concept of widespreadness. Widespreadness can be defined as the extent to which an attack has spread throughout the network. This value may not have as much importance as the danger and transferability values, but should at least reflect whether the attack has indeed spread away from a single host in the network. First impression is to add this as an additional factor in the determination of the Alert Level value. However, that would result in a much wider range of values over which to distribute the responses and would make it difficult to come up with meaningful values with respect to the response choices. It might make more sense to treat this as an independent variable with only two values: 1 for no spreading and 2 for having spread to more than one host. Then the appropriate response might be to notify all hosts if a value of 2 is reached. This value when combined with the Alert Level value might result in more stringent responses if this attack continues to spread than if widespreadness was not factored in at all.

3. Thresholds

One consideration is for the regulation of false positives and false negatives. The system does not want to be "crying wolf" all the time, but on the other hand, does not want to overlook any real attacks. Therefore, some form of threshold needs to be established against which to make a determination to take action or to send out notifications. One such mechanism for determining this might be to use the concepts of precision and recall. These values can be determined by observing the system behavior during some form of testing phase. The recall can be determined to be the value of the number of attacks that actually were recognized out of the number of attacks that actually occurred. The precision value can be measured by determining the number of true attacks out of those that were identified as attacks, as opposed to false alarms. Once good estimates of these values can be determined for each attack type, then they can be used to determine the threshold against which to take action. Increasing the threshold for an alert category usually decreases recall, but increases precision. So an expert could adjust the thresholds for the best tradeoff in their opinion or according to what policies he or his organization has set forth.

4. Network Identification

One of the most challenging parts of identifying attacks in a network is recognizing coordinated attacks. Often times an intruder may want to deceive traditional intrusion detection tools by distributing his attack across multiple sessions and/or multiple users. Each event in and of itself may not be recognizable as anything out of the ordinary. However, taken in aggregation, they may indeed be the components of a coordinated attack. An example of such an attack is the Doorknob attack. An intruder attempts to gain access to hosts in the network by trying various common account and password combinations on each host. Rather than sitting on a particular host and continuously repeating the attempt until successful or until a lockout occurs, he will try it a couple of times, move on to another host and repeat the attempt, etc. The intruder persists until he succeeds, much like a burglar walking down a hallway rattling the door of various apartments until he finds one unlocked. The rattles he makes are basically unnoticeable, so he is less likely to get caught, similar to the attempted logins. If an intruder keeps the number of attempts to a minimum on each host, this would fall into normal behavior and not be recognized. But if the aggregation of all these attempts were known, then it would be obvious that an attack was taking place.

In order to recognize attacks of this sort, some mechanism must be applied to recognize that each one of these attacks are coming from the same source, even if he has attempted to masquerade himself by changing user names. It might be difficult for each host to relay enough information to other hosts about the user so that these other hosts can associate an alert with a current user or attack taking place in its environment. It may be necessary to filter these alert messages through a separate Network identification tool that can resolve these users down to a common network identifier. Then, this tool can keep track of possible coordinated attacks.

Ko et al. [Ref. 5] has devised an algorithm for distributed recognition and accountability that could prove useful for performing this association. Such an algorithm could be run on a separate host to perform analysis to recognize coordinated attacks and send out alerts to affected hosts as necessary. Also, the various hosts that generate their own alert messages could filter these through this tool to see if a particular attack can be associated with any suspected ongoing coordinated attacks. Then a network identifier could be attached to any further alert messages to further keep track of a particular intruder's activity.

5. Communications

In order for the agents to communicate, a communication protocol must be established. The main reasons for providing a communication mechanism between agents is to provide for preemptive response to intrusions and to perform cooperative recognition of certain types of attacks. Certainly, each host can respond to a potential attack it discovers in its own environment, but once an attack is discovered, the damage may be done. It would be better to respond to potential attacks before they occur. If a system could be notified by an alert that the potential for an attack is imminent, then the system might be able to take some protective measures beforehand. Therefore, it is necessary for the nodes in the network to notify each other of potential attacks via alert messages containing information about the level and type of alert as described above in the section on Alert Levels. Each agent could then be configured to take whatever preemptive response it deems necessary, including specialized activity logging or lowering the thresholds against which to respond.

Also, there are certain types of attacks, such as doorknob rattling or sweep attacks that would be difficult to recognize on a single host. With the absence of a network monitor in this system, but which could be added at any time due to the extensible nature of the system, these types of attacks might go unnoticed. However, if an agent receives an alert suggesting a potential attack of this type, then its existence could be confirmed by the existence of further evidence on the machine that received the alert. Eventually, one of these agents might send out an alert confirming the attack, at which point all the nodes in the network could respond immediately to the threat.

How often and under what conditions messages are sent can be configured on each host when defining the appropriate response to events. Generally, new messages may be generated upon the change in Alert Level of an attack or upon the introduction of new intrusive activity. If messages are generated whenever something new is detected, then this could result in an overload of messages, particularly to busier machines. So, new messages should only be generated when the new intrusive activity has reached a particular threshold that is meaningful for that attack type. This concept has already been discussed in the responses associated with the various Alert Level ranges as described in

Section 2 of this chapter. The number of messages generated and to which machines are a function of the Alert Level.

One obvious problem associated with communications between the hosts is availability. If a host is unavailable to receive or send messages, it degrades the effectiveness of the intrusion detection system. Reasons for unavailability vary from planned downtime to being a victim of a denial of service attack. The former does not present any problem as the host is not subject to any compromise to an attack at that time, but the latter does indeed present a problem and knowledge of this attack needs to be made known to the other hosts in the network. If the host is unavailable, then it might not be able to send any messages notifying others of the ongoing attack. Also, if the host becomes unavailable for more benign reasons, it still needs to receive and send messages concerning other intrusion attacks. Therefore, the relative importance of sending and receiving message should increase with the increase in load on a host. This would make sure that, at the very least, messages will transfer properly.

However, it is possible that a host could become so overloaded that it is completely unavailable to send and receive messages and that fact needs to get out to raise the Alert Levels in other hosts in the network so they can protect themselves against a potential denial of service attack, even if it is eventually discovered that an attack of this type was not the true cause of the unavailability. This can be done by having all computers periodically send "I'm still running" messages to one another so that the absence of one of these messages would act as a positive message indicating a potential denial of service attack and result in raising the Alert Levels of other computers with regard to this kind of attack. The frequency of these messages should not be too great so as to add an unnecessary increase in the message traffic.

In addition, there may be other types of messages that are broadcast. Certainly, messages indicating a change in alertness state are needed. But additional ones may be necessary, such as changing configuration settings or product upgrades, adding additional hosts to the host file, or requesting additional information from hosts, all dependent on the actual implementation of the various methods in the agents. A security feature must also be added to this protocol to protect this system from an attack itself, such as an encrypted identifier. The following illustrates a proposal for the information contained in an alert message between agents.

1) Generating Host: Identifies the host machine generating the message and upon which the suspected attack is occurring.

2) Alert Level: The alert level value as determined by the product of the Danger and Transferability.

3) Attack Type: Identifies the attack object class.

4) Network ID: The network identifier for the source of the attack if that can be determined. It should probably contain some sort of type identifier to indicate at what level this identifier is being generated, that is, either from a host monitor of from a separate network identification tool.

5) Security Code: An encrypted identifier (such as a PGP key) that would ensure that this message is coming from a host within the system and is not being spoofed.

Various designs could be made to enact this protocol. Certainly a broadcast type would be the most useful. But on a TCP/IP network, it would require a point to point protocol. Therefore, a host name file probably needs to be kept and maintained by each host in the network so that the agent would know to which machines to send the messages.

6. Agent design

As new intrusions are discovered and documented, their handling needs to be added to the system with a minimum of interruption. Ideally, the system need not go down at all, but only the relevant piece be updated or added. This is possible in an object-oriented design where the agent software is broken down into component modules. The interfaces between the modules remain the same, but only the implementations of the methods need to be changed. So, if the handling of a particular attack changes over time, only the algorithm written to handle the attack in that particular attack module need be changed without affecting the rest of the system. Figure 8 provides an example of the object model of the agent design using the OMT methodology notation. [Ref. 10] The forks below the rectangles represent inheritance.

This object model is likely incomplete, most likely lacking all the necessary attributes and methods to make it fully functional. This would have to be completed in a full fledged development process, which has not been applied here. The following offers more detailed descriptions of each object class:

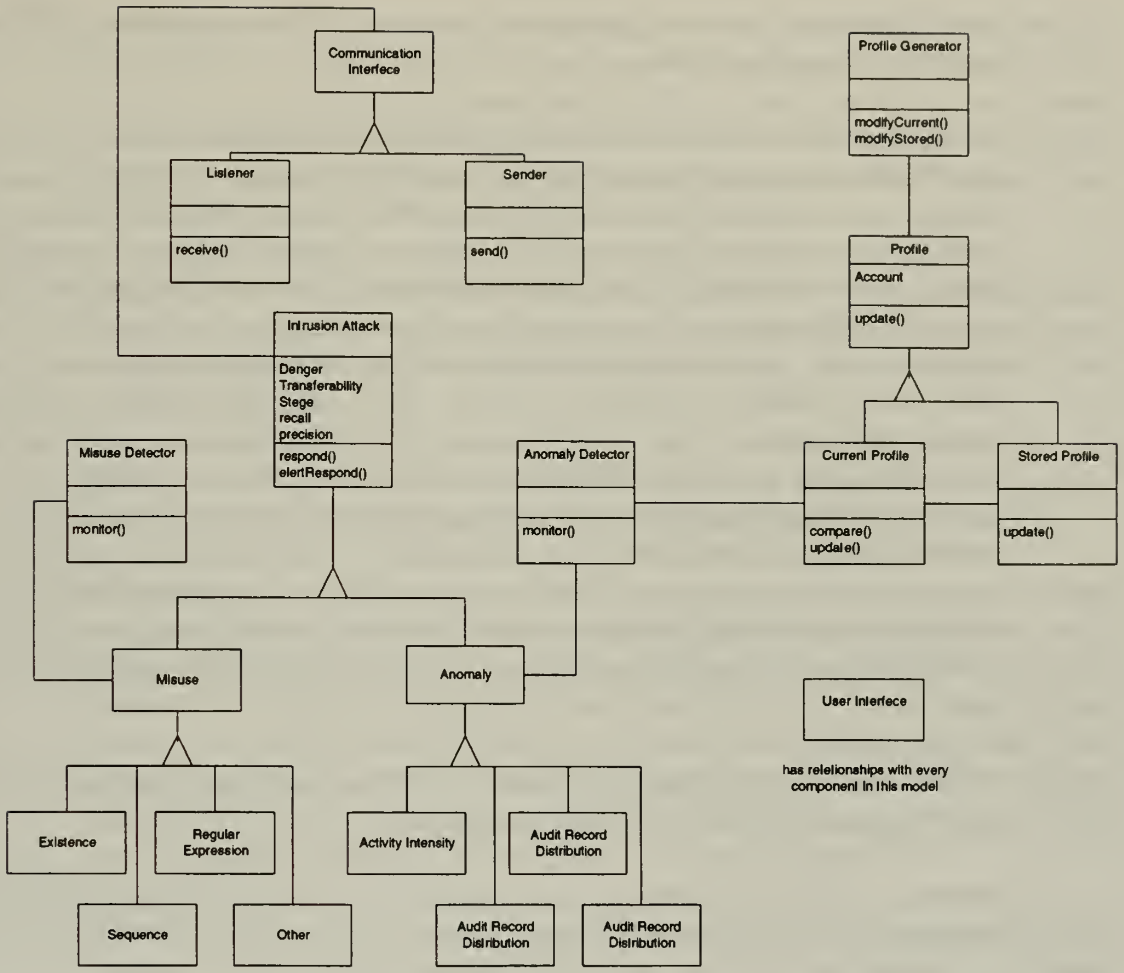

Figure 8. Object Model of Autonomous Agent Design

1) Communication Interface: This is an abstract superclass for the two types of communications that will occur, i.e., receiving or sending messages. It is not meant to be instantiated.

2) Listener: This is a subclass of Communication Interface. This class represents the piece of the software that listens for and receives incoming messages from other agents. It then processes them according to the type of attack described in the message by calling the class method alertRespond() on the Attack object class.

3) Sender: This is a subclass of Communication Interface. This class represents the piece of software that sends messages (alerts) to other agents. It knows

how to package the alert, identify the locations of other agents, and call the network software to send the message.

4) Intrusion Attack: This is the abstract superclass for all attack types. It is in here that the common attributes and methods for all intrusion types are placed. danger and transferability are attributes used to determine the alert levels of notification messages and whose values can possibly change, especially danger, with variations in the stage of an attack. The stage attribute is an internal attribute used to keep track of the various stages of an attack. It is the changes in this value that would trigger changes to the danger value, which, in turn, would change the alert level for that attack and possibly trigger additional responses. It also would provide visibility to an administrator to the current stage of an attack. Recall and precision are likely to be static values set by administrators regulating the threshold for responses. This may be redundant as recall and precision are functions of one another and only one value may need to be stored. The method respond() is the method defined to respond to an internal change in alert level and alertRespond() is a static method defined to respond to receipt of alert level messages for attacks of that type.

5) Misuse: This is a further specialization of Intrusion Attack that covers all attack types that fit into the misuse category.

6) Anomaly: This is a further specialization of Intrusion Attack that covers all attack types that fit into the anomaly category.

7) Attempted Break in, Activity Intensity, etc. These are the finer grained subclasses of Misuse and Anomaly that specify all the various attack types. It is here that the specific handling requirements are implemented, overriding any general implementations that may have been implemented at the superclass level. It is intended that the number and identity of these classes be dynamic and new ones added as needed. It is possible a class at this level may indeed represent a generalization to further subclasses, depending on the attack type defined. Objects at this level will communicate with the Listener portion of the Communications Interface to pass it information needed to create an alert to send to other agents.

8) Misuse Detector: This is one of the main components of the agent software. It will be responsible for monitoring the system resources and audit files to seek out and identify possible attacks. If it finds one, it will instantiate an object of the appropriate class and maintain its state and call the appropriate method to handle the suspected intrusion.

9) Anomaly Detector: Sister to the Misuse Detector. It will be responsible for monitoring the system resources for anomalous behavior. It calls the Profile of various accounts to compare itself to the stored Profile. If the current profile is differing significantly, then anomalous behavior is occurring. It will then instantiate an object of the appropriate class and maintain its state and call the appropriate method to handle the suspected intrusion.

10) Profile Generator: This represents another workhorse of the agent software. It continuously monitors the system resources and audit files to maintain the state of the current profile and to periodically merge the current profile into the stored profile.

11) Profile: This class represents an abstract superclass to two types of profiles, the current profile of a system account, and the stored profile of an account.

12) Current Profile: This class represents the current profile of a particular system account. Current implies that it represents only the profile generated for some determined period of time up to and including the current time. That length of time can be variable, depending on what value best returns a valid profile that can be compared against the stored profile with some confidence. This class is called by the Anomaly Detector to compare itself to the stored class and return information specifying whether the current behavior has deviated enough from the stored profile to represent a possible attack.

13) Stored Profile: This class represents the stored profile of a particular system account. This is the historical, statistical record of an account's behavior against which the current profile can be compared to determine whether the current behavior is anomalous. This profile needs to be updated by merging in the current profile data at periodic points, probably whenever the current profile is updated with new data itself.

Each of these agents is responsible for monitoring the activity on a single host in the network seeking to identify single-host attacks as well as components of network-wide attacks occurring on that host. This monitoring occurs within the Misuse and Anomaly Detectors. Once an attack has been identified, an occurrence of it is instantiated from the attack class hierarchy and maintained by that agent through continued monitoring and through coordination with the other agents in the system through communication via alert level messages. When an agent receives an alert level message via its Communication Interface, it calls a static method called alertRespond() belonging to the attack class relevant to that alert level message. It is in this method where the decision making occurs on how to respond to receiving that message. Preemptive response would be implemented here as well as changing the state of relevant attack objects that exist on that host.

7. Example Intrusion Scenario

To demonstrate an example of how this design might be put to use, I will describe a potential implementation and illustrate how an attack might be handled. In this example I will illustrate the identification and response to a *Doorknob Rattling* attack in a network. This particular attack will be coordinated; that is, a simultaneous sweep from multiple sources. This utilizes two features enabling the intruder to stay hidden. First, the sweep portion allows the intruder to not raise suspicion at any given period of time and the multiple sources hinders the recognition of an association between simultaneous events. Figure 9 shows the layout of a small, sample network.

In this example, Host 1 and Host 2 are UNIX platforms running an autonomous agent, while Host 3 is a Windows NT platform running an autonomous agent. Host 4 is a UNIX platform running a network monitor. Each of these hosts communicate with each other using the sockets mechanism provided with the operating system using the TCP/IP protocol to transfer the messages. The listener is a daemon process that retrieves messages coming into a TCP/IP port on a host. This listener passes the information in the message to the communication interface in the agent software.

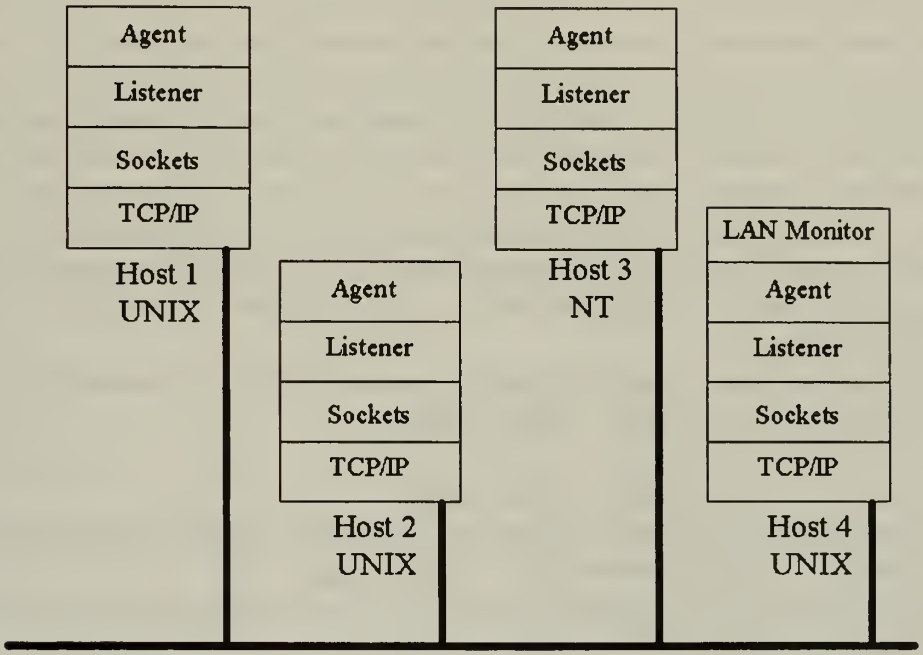

Figure 9. Example Network Subjected to Doorknob Attack

Each host with an agent is configured identically for handling of doorknob attacks; that is:

1) Respond to notifications of this attack type with alert level values of 6-10 by monitoring logins more closely (run a special process to do this) and accumulate evidence.

2) Respond to notifications of this attack type with alert level values of 11-15 by paging the system administrator.

3) Respond to continued attempts on that host, at least six, while alert level is in a state of 6-10 by raising the danger level to three.

4) Respond to a successful illegal login on that host by raising the danger level to a value of four.

5) Relevant operating systems are UNIX and Windows NT.

While not shown in detail in this example, it is assumed that precision and recall values have been previously determined to set the thresholds as just specified. The host with the monitor is configured to respond to an identification of a potential doorknob attack by setting the danger level to a value of two. Each host has a host file (could be a single file on a shared drive) with information about the other hosts in the network that looks like the following:

1) Host_1: OS: UNIX Type: Agent

2) Host_2: OS: UNIX Type: Agent

3) Host_3: OS: Windows NT Type: Agent

4) Host_4: OS: UNIX Type: Monitor

An attack scenario in this environment might go as follows. The LAN monitor on Host 4 recognizes that multiple attempts to login to various hosts are originating from the same source. An instance of a Doorknob Attack subclass in the Intrusion Attack hierarchy (not shown in Figure 8, but presumed to be part of the hierarchy as defined for this system) is created by the agent's Misuse Detector interfacing with the LAN Monitor on the same host. The Misuse Detector knows when to create this instance via its interfacing with the LAN Monitor. It then determines that the danger level of the newly created Doorknob Attack object has a value of two, since it is not absolutely positive this is an actual attack and the system has been configured respond this way. Then the Misuse Detector determines which machines would be affected. In this case, it would be all of the machines, as both UNIX and Windows NT operating systems have login functions. The Misuse Detector then sets the transferability value on the Doorknob Attack object to three since all machines in the network are being affected, indicating an Alert Level of six. This implies partial notification, meaning only those machines which are relevant need to be notified. The Misuse Detector then calls the respond() method of the Doorknob Attack object and this method causes notification to occur by sending messages to all the hosts in the host file that are relevant; that is, whose operating systems are UNIX or Windows NT. But in this case, all the nodes are relevant, so they all will be notified.

When each host gets the notification, it responds to that Alert Level based on how the response was configured for that host. In this example, all the hosts are configured identically. So, the message is received via the socket mechanism residing on a network port of each host machine. This socket forks a process to handle receipt of that message. This process passes the information in the message to the Listener object in the agent via its receive() method. The Listener object then calls the alertRespond() static method of the Doorknob Attack class to handle the response to this alert message. The response in this case will be for each host to monitor the logins on itself more closely to determine if further logins continue to come from the same source and to start accumulating evidence of this attack. Host 1 sees six more attempted logins on its machine and therefore raises the danger level to three since it was configured to respond in this way. The attack is almost surely determined, but the seriousness of the consequences are not great. Just logging in causes no damage if stopped there. The transferability level remains the same.

So, the Alert Level now is raised to nine. Messages are sent out from Host 1 to Host 2 and Host 3. However, Host 2 recognizes an actual successful login from this source and the danger level is raised to four to indicate "very serious." With the Alert Level now at twelve, it is placed into the highest category, meaning that all hosts will receive a notification. So, Host 2 sends a notification to Host 1 and Host 3 and immediately takes action to page its administrator. When Host 1 and Host 3 get a notification with an Alert Level of 12, they too page their respective administrators. The administrators would then take whatever action they deem necessary. This would end the current scenario.

Alternatives exist for some of the actions in this scenario. Receipt of the notification at an Alert Level of 12 might have been handled by sending a notification to a firewall to filter out all packets coming from this source. Or it might be sent to a tool to start tracing the actual source of the attack. This system can be extended or modified at any time by changing the configurations at each host to reflect differing approaches in responding to events.

C. SUMMARY

The Distributed Autonomous Agent Network Intrusion Detection and Response System, as just proposed, meets, at least in some portion, all the requirements as stated in Chapter II outlining a good network intrusion detection system. There is much more work to be done in applying an actual implementation, but the architecture and design offered here provide a solid basis. In addition to providing a conclusion to this thesis, the next chapter will review these requirements as stated in Chapter II and provide details as to how this system meets them.

V. CONCLUSIONS AND RECOMMENDATIONS

A. REQUIREMENTS SATISFACTION

The following reviews the requirements for a good intrusion detection system as stated in Chapter II and relates how the above design satisfies these requirements.

1. Requirement 1: Recognition

The system must recognize initial potentially suspect behavior (the triggering event): The Misuse Detector and the Anomaly Detector are the components that seek and recognize suspicious behavior. They exist as part of every agent that sits on every host. In addition, the central collection of some of the activity on various hosts allows for the recognition of distributed attacks.

2. Requirement 2: Escalating Behavior

The system must recognize escalating suspicious behavior on a single host or across hosts: The Misuse Detector and the Anomaly Detector are both used to continuously seek and recognize escalating behavior. In addition, the attack object classes allow for the instantiation of particular attack threads that have a life cycle for as long as the threat is active.

3. Requirement 3: Communication

The system components on various hosts must communicate with each other apprising them of a change in the level of alertness whether escalating or de-escalating: A communications protocol was established containing the information necessary to relate the alert levels and attack types from one host to another. The Communications Interface hierarchy in the object model also supports the generation of, sending to, receiving of, and response to these messages.

4. Requirement 4: Response

The system components must take the appropriate action that corresponds with a level of alertness: The attack class hierarchy contains a response method that is defined for each attack class. The actual implementation should be written to respond according to the level of alertness, attack type and stage of attack.

5. Requirement 5: Manual Control

The system must allow for manual intervention in the changing of alertness: The User Interface allows for direct manipulation of each object in the model. This gives the system administrator direct control over the state of any object in the system. Furthermore, each agent can act as the master to the system at any given time. This would allow the system administrator access at any given node in the network. If an attack requires manual intervention, the system administrator can use any machine to perform the intervention and propagate the results to all the other relevant nodes.

6. Requirement 6: Adaptability

The system must be highly adaptable to be able to respond quickly to the ever changing methods of attack: The existence of an attack hierarchy allows for the addition of new attack types with ease. Therefore, the system can respond as quickly as it takes to write or modify an algorithm to respond to an attack. Since each response is kept in separate attack classes, these modifications can be applied with no trickle down effects on other parts of the system. Therefore, the system can keep functioning properly with respect to attacks other than the one being added or modified. However, there is a downside to this. This system is adaptable in the sense that the modularity of the system makes it easier to add new attack types and responses without affecting the rest of the system, but not necessarily in real time. The system does rely on experts to analyze new threats and define its features and that can take months. However, in the few cases where a new threat can be analyzed immediately by an administrator or programmer, this system would support the addition of this attack class in near real time. Ideally, this system would adapt on its own. One aspect not addressed here was the ability to learn. A very sophisticated system could learn to identify, or be trained to handle, new attacks as they become apparent, thus making it more adaptable in real time. This is discussed further in the conclusions section of this thesis.

7. Requirement 7: Concurrency

The system must be able to handle multiple, concurrent attack threads: The architecture alone provides for this as each agent can deal with attacks that are occurring only on its own machine. However, the existence of the attack hierarchy allows for further concurrency by providing for the concurrent instantiation of attack objects. This means that whenever a unique attack is recognized, its existence is recorded in an instantiation of an attack class of that attack type. The Misuse and Anomaly Detectors can keep track of and communicate with each instantiated attack object. Each instantiation can be realized on the system as a separate process allowing for concurrent attack threads.

8. Requirement 8: Scaleability

The system must be able to scale as the network grows: The architecture provides for this. As a node is added to the network, so is an agent to run on that node. Since the agents run independently of each other, they can be added without adversely affecting the performance of the other agents. Unlike other systems that monitor network traffic at a single point in the network where the increase in data that results from growth might cause serious performance problems, independent agents only have to deal with the data for a single system. As new hosts are added to the network, the number of messages received by any one agent should increase linearly on average. But will the processing of these messages increase linearly? I think they would. I envision that each agent processes each message on a first come, first served basis. There would be no need to sort messages or nor should there be any relationship between messages. As each message is received by the listener on each host, the host forks off a process to handle it. Therefore, each host can handle messages in rapid succession. The amount of time to handle a message should be fairly short, so the total number of forked processes at any given time should peak at an amount that is based on the number of processes that can be forked within the average process time of a message. This should fall well within the capabilities of each machine. If a host is unable to receive a message, the sender can retry for some set number of time before quitting to prevent backlog of messages.

9. Requirement 9: Resistance to Compromise of Intrusion Software Itself

The system must be able to protect itself from unauthorized use or attack: The actual ability to protect itself depends on the implementation. There are many ways to secure software outside the scope of this thesis, but the system design offered here also supports efforts toward this end. The addition of an encrypted security key to each message provides protection from spoofed messages. And any attempt to access the system itself can be thwarted by the addition of an attack class of this type of attack. If the method of attack on the security system is known, then that attack can be modeled into an attack class and the system itself can monitor for any activity matching that model and respond to it appropriately. Furthermore, the heterogeneous nature of the agents makes it more difficult to define a single attack that could affect all the agents in the system with the same intended results, thus making it difficult to define an attack that could compromise the whole system.

10. Requirement 10: Efficiency and Reliability

The system should be efficient, reliable, resistant to degradation and run in a heterogeneous platform environment: The ability to apply the most effective detection method directly to each class of intrusion type makes it more efficient. Only the most appropriate code need be executed and only on the machine where the attack is taking place. If any node agent goes down, they system still runs in full. Only the machine upon which the agent was running becomes vulnerable. The rest of the network is protected. The distribution of the load also goes to support resistance to degradation. And each agent is implemented specifically for the platform upon which it runs, but can communicate with each other regardless of platform.

11. Requirement 11: Extensibility

The system must be extensible: As new attack types are defined, they can be added to the various agent implementations and only to those agents to which this type of attack is relevant Furthermore, this architecture can support the augmentation of additional intrusion detection or monitoring tools with little impact on the existing system. For example, LAN monitoring tools, network sniffers, combined audit trail analyzers can be added and communicate with the rest of the hosts in the network as if they were a host of their own.

12. Requirement 12: Flexibility

The system must be flexible in the establishment of security policy and management across hosts: The autonomous nature of the agents on each host that can perform any subset, up to the full set, of functionality of the system allows for this flexibility. Each administrator can configure his own values of recall or precision to establish his own thresholds against which to perform some action. In addition, he can choose which attack types to monitor, or even write his own kind of responses. In a network that crosses organizational boundaries, this is very important because it may be difficult to enforce a centralized policy and administrators may be hesitant to even install this system on their host. Some policy is better than none.

B. CONCLUSIONS

This thesis has proposed an architecture and design of a real-time intrusion detection and response system for use in large heterogeneous networks. The field of applying intrusion detection techniques that can be effective in recognizing intrusions in a network and then responding to these threats in real time is relatively new. The number of robust, yet efficient solutions is limited to a few. This thesis proposed a solution that attempts to satisfy all the major issues of a network-wide intrusion detection system; namely, scaleability, flexibility, extensibility, fault tolerance, efficiency and, of course, ability to perform.

A distributed architecture composed of autonomous agents and an object-oriented design were used as a means to this end. Most intrusion detection systems to date were just that. They made little attempt to include real-time response to intrusions. While intrusion detection is valuable, detection after the fact could come too little too late. Ideally, an intrusion detection system can respond to an imminent attack to mitigate the potential damage, even if the response is simply to page an administrator so that a manual response can be performed. The system proposed in this thesis provides for any number of automated responses to different types of attacks with varying levels of seriousness.

Another level of response not addressed in great detail in previous systems is performing preemptive measures to protect a host against imminent attacks as a result of recognizing the existence of an attack somewhere else in the network. This thesis provides a solution for this as well through the use of alert-level notification messages.

C. RECOMMENDATIONS

Not all issues were resolved. A system of this type requires an agent to be running on every host that wishes to be protected. However, any host that does not install one of these agents will not be adequately protected. It might receive some protection if a network layer monitor is added as a component, but that is the case only if there is some network level response, such as denying access to the network from the source of the attack through a firewall or such. If this is not done, then any host without an agent is still vulnerable to attack since it cannot respond to an attack or notifications of a potential attack.

Additionally, while this system can be used to identify known misuse attacks or anomalous behavior, it does not yet adequately deal with recognizing misuse attacks not previously documented. New kinds of attacks are appearing everyday. Unless the administrators of this system are constantly updating the attack profiles, the network will always be vulnerable to unknown attacks. It would be ideal if mechanisms could be developed that could learn to recognize new attacks, especially if they are just variants of known attacks, and automatically apply them. This will not be an easy task.

Also, while the concepts of recall and precision were proposed in this thesis to deal with the issue of false positives and negatives, it would be desirable to have a way to minimize the number of false positives and negatives instead. Crosbie and Spafford [Ref. 8] proposed a training module with their system that would teach the agent how to better recognize attacks using various scenarios and using genetic programming. This kind of approach could be useful to supplement the architecture as proposed in this thesis.

This thesis proposed an architecture and design. No implementations have been provided. In fact, the architecture and design is meant to be open so that implementations can be swapped out at any time when newer, better approaches become available thus permitting the system to grow with technology and avoid the problem of becoming obsolete. Further work would be required to move this system from the design proposed here to a final working product.

LIST OF REFERENCES

1. Aurobindo Sundaram. *An Introduction to Intrusion Detection,* http://www.techmanager.com/nov96/intrus.html

2. Mark Crosbie and Gene Spafford. *Defending a Computer System using Autonomous Agents.* Technical Report No. 95-022, COAST Labratory, Dept. of Computer Sciences, Purdue University, March 11, 1994

3. Sandeep Kumar and Eugene H. Spafford. *An Application of Pattern Matching in Intrusion Detection,*
Technical Report No. CSD-TR-94-013, The COAST Project, Dept. of Computer Sciences, Purdue University, June 17, 1994

4. M. Sobirey, B. Richter and H. König. Adaptive Intrusion Detection System, http://www-rnks.informatik.tu-cottbus.de/~sobirey/aid.e.html

5. Calvin Ko, Deborah A. Frincke, Terrence Goan Jr, L. Todd Heberlein, Karl Levitt, Biswanath Mukherjee and Christopher Wee. *Analysis of An Algorithm for Distributed Recognition and Accountability.* 1st ACM Conference on Computer and Communications Security, Dept. of Computer Science, University of California, Davis, November 1993

6. Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. *Network Intrusion Detection.* IEEE Network, 8(3):26-41, May/June 1994

7. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle. *GrIDS - A Graph Based Intrusion Detection System for Large Networks*, Proc. of the 19th National Information Systems Security Conference, Baltimore, MD, Oct. 1996, 361 - 370

8. Mark Crosbie and Gene Spafford. *Active Defense of a Computer System using Autonomous Agents.* Technical Report No. 95-008, COAST Group, Dept. of Computer Sciences, Purdue University, February 15, 1995

9. Sandeep Kumar. *Classification and Detection of Computer Intrusions.* Department of Computer Sciences, Purdue University. PhD Dissertation, 1995.

10. James Rumbaugh, *Object-Oriented Modeling and Design.* Pretence Hall International, New Jersey, 1991

BIBLIOGRAPHY

The following lists several resources that were very helpful in preparing this thesis, but were not actually cited in the text

Frank, Jeremy. *Artificial Intelligence and Intrusion Detection: Current and Future Directions.* Division of Computer Science, University of California at Davis, June 9, 1994

Ilgan, Koral. *USTAT, A Real-time Intrusion Detection System for UNIX.* University of California, Santa Barbara. Master's Thesis, November 1992

Kyas, Othmar. *Internet Security: Risk Analysis, Strategies and Firewalls.* International Thomson Computer Press, 1997

Stoll, Clifford. *The Cuckoo's Egg.* Doubleday 1989

INITIAL DISTRIBUTION LIST